

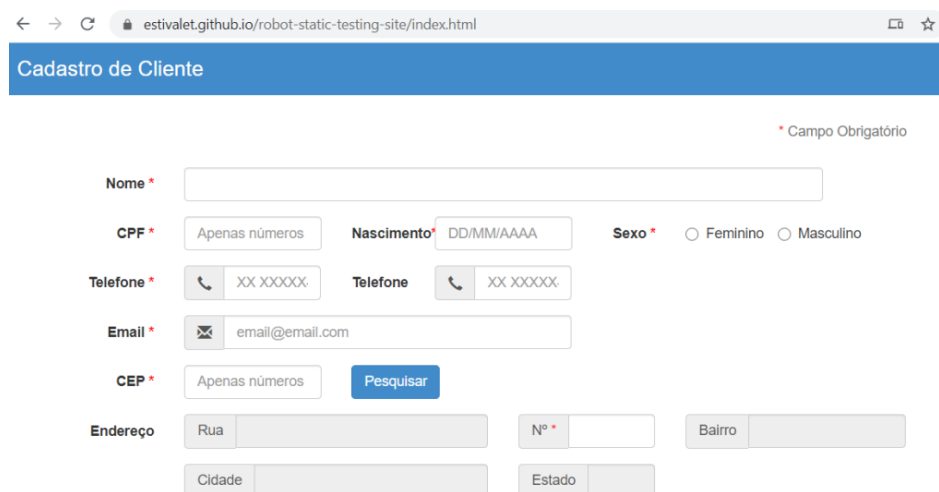
U Learn Programming

Teaching computer languages for everyone

Testes Automatizados Usando Robot Framework – Módulo 1 – Automatizando uma página de cadastro de clientes

Vamos agora automatizar uma página de cadastro de clientes. Essa página poderia ser um módulo de sistema real, mas vamos usá-la aqui no curso apenas para fins de ensinar como tratar os diferentes elementos de tela com o Robot Framework.

A página está localizada [aqui](#).



The screenshot shows a web browser window with the address bar displaying 'estivalet.github.io/robot-static-testing-site/index.html'. The page title is 'Cadastro de Cliente'. The form includes the following fields and elements:

- Nome ***: A text input field.
- CPF ***: A text input field with a placeholder 'Apenas números'.
- Nascimento ***: A text input field with a placeholder 'DD/MM/AAAA'.
- Sexo ***: Two radio buttons labeled 'Feminino' and 'Masculino'.
- Telefone ***: Two text input fields, each with a placeholder 'XX XXXXX' and a telephone icon.
- Email ***: A text input field with a placeholder 'email@email.com' and an email icon.
- CEP ***: A text input field with a placeholder 'Apenas números' and a blue button labeled 'Pesquisar'.
- Endereço**: A group of five text input fields: 'Rua', 'N°', 'Bairro', 'Cidade', and 'Estado'.

A legend indicates that an asterisk (*) denotes a required field ('* Campo Obrigatório').

Cadastro de clientes

Como podemos perceber existem diversos campos para entrada de dados: campos de texto, data, caixas de seleção entre outros.

Vamos criar um arquivo `cadastro-cliente.robot` com o seguinte conteúdo:

*** Settings ***

Library SeleniumLibrary

*** Test Cases ***

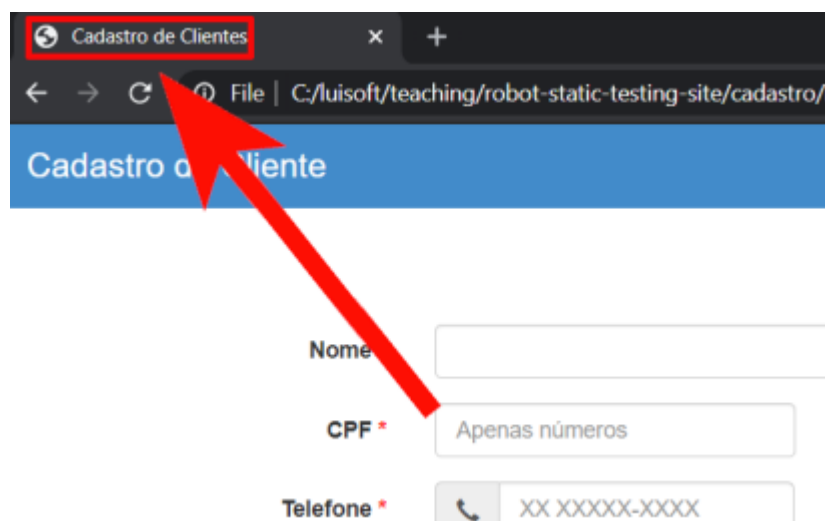
Fazer o cadastro de um novo cliente

Open Browser <https://estivalet.github.io/robot-static-testing-site/cadastro/index.html> chrome

Title Should Be Cadastro de Clientes

Close Browser

Apenas para relembrar, estamos fazendo uso da biblioteca **SeleniumLibrary** para poder interagir com o navegador. Criamos um caso de teste chamado **Fazer o cadastro de um novo cliente**. Usamos a *keyword* **Open Browser** para abrir a página de cadastro de clientes e estamos especificando que o navegador utilizado será o Google Chrome. Uma primeira validação que podemos fazer é verificar se o título da aba é **Cadastro de Clientes**. Para isso usamos a *keyword* **Title Should Be**.



Título da aba

Vamos rodar o robô abrindo um terminal na pasta onde o arquivo está salvo e digitamos o comando: “**robot cadastro-clientes.robot**” sem as aspas. O robô vai iniciar a execução do teste abrindo o Google Chrome e navegando até a página de cadastro de clientes.

Ao final da execução no terminal devemos ter o seguinte resultado.

```
PS C:\luisoft\teaching\robot-module-1> robot .\cadastro-clientes.robot
=====
Cadastro-Clientes
=====
Fazer o cadastro de um novo cliente
DevTools listening on ws://127.0.0.1:57069/devtools/browser/5e001945-b9f1-4f27-88e3-d6b44cd767ec
Fazer o cadastro de um novo cliente | PASS |
-----
Cadastro-Clientes | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
=====
Output: C:\luisoft\teaching\robot-module-1\output.xml
Log: C:\luisoft\teaching\robot-module-1\log.html
Report: C:\luisoft\teaching\robot-module-1\report.html
```

Nosso teste passou. Para confirmar que está tudo certo podemos alterar o texto na *keyword* **Title Should Be** para algo diferente para “Robot Framework” e rodar nosso teste novamente. Dessa vez veremos o teste falhar.

```
Fazer o cadastro de um novo cliente | FAIL |
Title should have been 'Robot Framework' but was 'Cadastro de Clientes'.
-----
Cadastro-Clientes | FAIL |
1 critical test, 0 passed, 1 failed
1 test total, 0 passed, 1 failed
=====
```

O Robot Framework parou a execução na *keyword* **Title Should Be** exibindo o erro “Title should have been ‘Robot Framework’ but was ‘Cadastro de Clientes’”, ou seja, está dizendo que o título deveria ser “Robot Framework” mas o que foi encontrado foi “Cadastro de Clientes”. Claro que fizemos isso apenas para forçar o teste falhar. Vamos voltar o texto para “Cadastro de Clientes” para que o teste volte a passar.

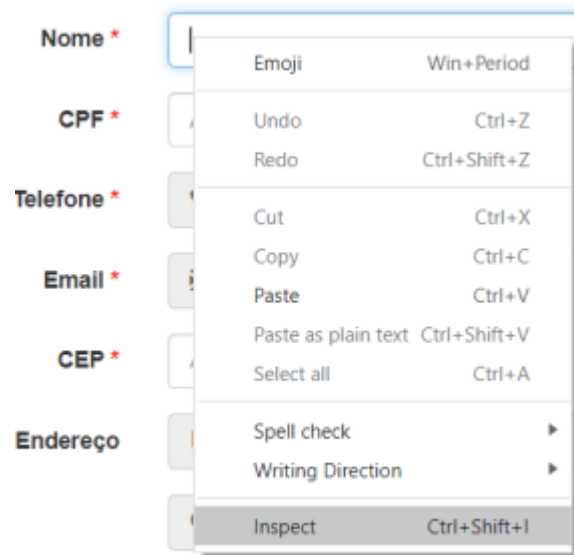
Preenchendo os campos do formulário

Campos de texto

Para o Robot Framework poder manipular os campos da página é necessário informar a forma de localização desse elemento. Existem diversas formas para [localizar um elemento](#) mas as mais comuns são por: **id**, **name**, **css** e **xpath**.

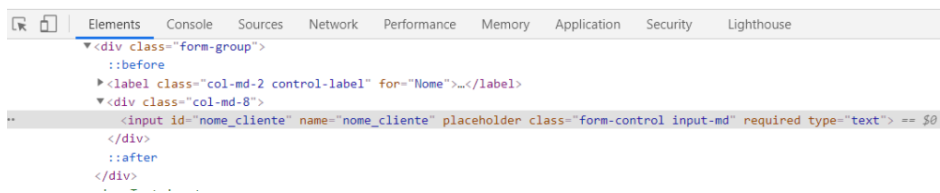
Vamos localizar o primeiro campo do formulário, o nome do cliente. Para fazer isso podemos usar o próprio Chrome. Com a página do formulário de cadastro de clientes aberta clique com o botão direito do mouse dentro do campo nome e selecione a

opção “**Inspeccionar**”. Nesse momento será aberta a ferramenta de desenvolvimento do Chrome.



Clicando com o botão direito do mouse e selecionar a opção de inspeção de elemento

E na aba “**Elementos**” é mostrado os detalhes do campo nome do cliente:



Como podemos ver é um campo do tipo *input*, ou seja, de entrada de dados. Nesse caso ele tem as propriedades **id** e **name** o que torna mais fácil a identificação desse elemento. Vamos usar o “**id=nome_cliente**” para informar um nome para nosso cliente.

A *keyword* que o Robot Framework usa para preencher um campo de texto é [Input Text](#). Vamos adicionar essa *keyword* no nosso caso de teste.

*** Settings ***

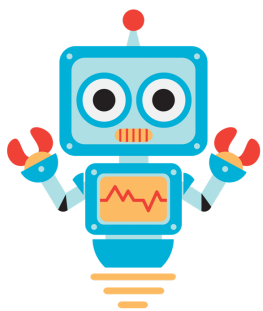
Library SeleniumLibrary

*** Test Cases ***

Fazer o cadastro de um novo cliente

Open Browser	https://estivalet.github.io/robot-static-testing-site/cadastro/index.html	
Title Should Be	Cadastro de Clientes	
Input Text	id=nome_cliente	José da Silva

Vamos rodar novamente nosso teste, dessa vez notem que retiramos a **keyword Close Browser** para deixar o navegador aberto e ver que o Robot Framework realmente colocou o “José da Silva” no campo nome do cliente.



DESAFIO

*Agora tentem fazer vocês o mesmo para os dois próximos campos do nosso formulário: CPF e data de nascimento. Será necessário identificar os elementos e adicionar novas keywords **Input Text** para colocar um texto qualquer nesses campos.*

Botões de seleção

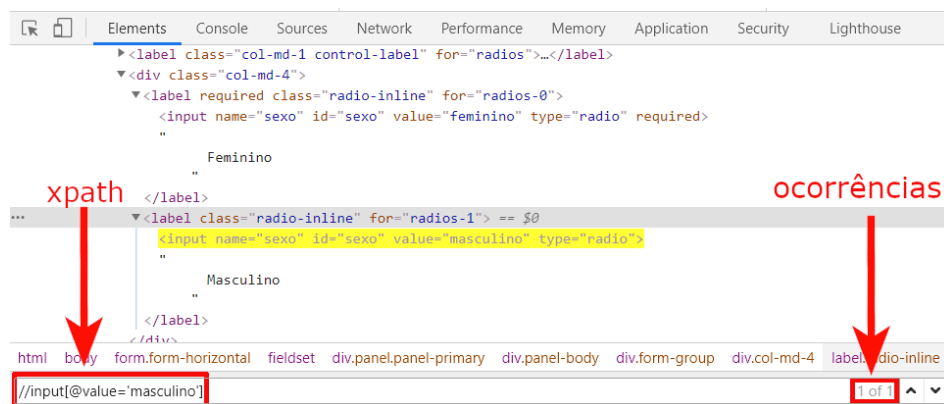
Vamos ver como tratar os *radio button* (botões de seleção) com o Robot Framework. Precisamos escolher o sexo do cliente que estamos cadastrando. Porém agora temos um problema! Não vamos poder usar o “**id**” pois ambos são iguais para os dois valores, masculino e feminino.

```
▼<label required class="radio-inline" for="radios-0">
  <input name="sexo" id="sexo" value="feminino" type="radio" required>
  "
    Feminino
  "
</label>
▼<label class="radio-inline" for="radios-1">
  <input name="sexo" id="sexo" value="masculino" type="radio">
  "
    Masculino
  "
</label>
```

Tanto masculino quanto feminino tem o mesmo “id”

Nessa caso podemos recorrer ao localizador **xpath** para identificar unicamente o elemento que queremos clicar. O **xpath** é uma linguagem para selecionar nodos de um arquivo XML. Como as páginas do nosso sistema são em HTML então também podemos usar o **xpath** para selecionar o elemento desejado.

Podemos testar um **xpath** na própria ferramenta de desenvolvimento do Chrome. Estando na aba “Elementos” pressiona-se CTRL+F para procurar um determinado elemento. Será aberto um campo de texto para que possamos digitar um **xpath** para localizar um elemento. Vamos digitar o **xpath** “//input[@value='masculino']” sem as aspas. Esse **xpath** está procurando um **input** que tenha um atributo igual à “masculino”.



Podemos observar no canto direito é informado o número de ocorrências que foram encontradas usando esse **xpath**. Nesse caso foi apenas 1 e esse é o elemento que queremos interagir e portanto esse **xpath** pode ser utilizado no Robot Framework para clicarmos *radio button* sexo masculino.



Para que o Robot Framework possa interagir corretamente com o elemento de tela ele deve ser identificado de forma única, pois se tivermos mais de um elemento o Robot pode não estar trabalhando com o elemento correto.

Caso o número de ocorrências fosse maior do 1 então teríamos que escrever um **xpath** um pouco mais específico até que consigamos encontrar apenas o elemento que queremos interagir.

Vamos ter um curso para estudar melhor o **xpath**.

Agora que já temos como identificar o *radio button* sexo masculino, vamos clicar nesse elemento usando o Robot Framework. Para tanto, vamos usar a keyword [Click Element](#) e vamos passar o nosso **xpath**.

```
*** Settings ***
```

```
Library                               SeleniumLibrary
```

```
*** Test Cases ***
```

```
Fazer o cadastro de um novo cliente
```

```
    Open Browser                      https://estivalet.github.io/robot-static-testing-  
site/cadastro/index.html    chrome
```

```
    Title Should Be                  Cadastro de Clientes
```

```
    Input Text                       id=nome_cliente    José da Silva
```

```
    Click Element                    xpath=//input[@value='masculino']
```

Botões

Agora vamos preencher o campo de cep e clicar no botão pesquisar para completar os demais campos de endereço do cliente.

O campo cep é um campo de texto e já aprendemos como usar no Robot Framework. Vamos adicionar a keyword **Input Text** com **id=cep**.

Já o botão “Pesquisar” cep vamos usar a mesma *keyword* que usamos para manipular o *radio button*, a **Click Element**. Como só temos um botão com o texto “Pesquisar” na nossa página, podemos usar “**//button[text()='Pesquisar']**” que é o **xpath** que identifica unicamente esse elemento na página.

Logo após clicarmos no botão de pesquisa de cep queremos aguardar que os demais campos de endereço do cliente sejam preenchidos. Como esse botão dispara um serviço de pesquisa de cep, pode ser que demore um pouco para esse resultado retornar.

O Robot Framework provê diversas *keywords* para lidar com essa situação. Poderíamos determinar um tempo fixo, parando a execução do teste por 2 segundos usando a *keyword* [Sleep](#) mas essa não é uma maneira eficiente pois caso o serviço demore mais que 2 segundos para retornar correremos o risco do teste falhar e se for menos que 2 segundos vamos estar com o teste parado mesmo já tendo os dados necessários para continuar a execução.



Evite o uso da keyword Sleep ao máximo. Prefira sempre usar as keywords Wait da biblioteca do Selenium para deixar o seu cenário de teste mais eficiente. Talvez demore um pouco para achar a condição correta para esperar determinado elemento mas vale à pena.

A biblioteca do Selenium oferece diversas *keywords* do tipo **Wait** para aguardar até que uma condição seja satisfeita para o teste continuar. No nosso caso poderíamos usar a *keyword* [Wait Until Element Contains](#) que espera até que o elemento contenha um determinado texto, porém precisamos testar se o **valor do campo** foi preenchido com o texto, ou seja, queremos garantir que o atributo **value** do campo **rua** esteja preenchido corretamente ao final da pesquisa.

Infelizmente o Robot Framework ainda não tem uma *keyword* que faça esse teste diretamente, então vamos criar um *keyword* para essa finalidade.

Nosso código vai ficar assim:

```
*** Settings ***
```

Library SeleniumLibrary

*** Test Cases ***

Fazer o cadastro de um novo cliente

Open Browser	https://estivalet.github.io/robot-static-testing-site/cadastro/index.html
	chrome

Title Should Be Cadastro de Clientes

Input Text	id=nome_cliente	José da Silva
------------	-----------------	---------------

Click Element `xpath=//input[@value='masculino']`


```
Input Text          id=cep      21842640
Click Element       xpath=//button[text()='Pesquisar']
Wait Until Keyword Succeeds  10s    200ms    Aguardar o valor "Rua Otávio de
Farias" no elemento "id=rua" estar presente
```

*** Keywords ***

```
Aguardar o valor "${valor_esperado}" no elemento "${elemento}" estar presente
    ${valor_atual}=    Get Element Attribute    ${elemento}    value
    Should Be Equal As Strings    ${valor_atual}    ${valor_esperado}
```

Criamos a *keyword* **Aguardar o valor “valor_esperado” no elemento “elemento” estar presente**. Ela é bastante simples, primeiramente pegamos o atributo **value** do elemento passado como parâmetro e armazenamos na variável **valor_atual**. Ou seja vamos estar pegando o **valor** do campo **rua**. Depois testamos se esse valor é igual ao valor esperado usando a *keyword* [Should Be Equal As Strings](#) para comparar os valores.

Queremos esperar somente o tempo necessário para que o campo **rua** seja populado, então vamos usar a *keyword* [Wait Until Keyword Succeeds](#) que aguarda até que uma *keyword* seja executada com sucesso. Para essa *keyword* vamos passar 3 parâmetros. O primeiro informa que vamos aguardar no máximo **10 segundos** para que a *keyword* que definimos retorne com sucesso. O segundo parâmetro chama nossa *keyword* a cada 200 milisegundos. E por fim o terceiro parâmetro é a *keyword* que definimos. Ou seja a cada 200 milisegundos será verificado se o campo **rua** contém o valor **Rua Otávio de Farias**. Caso se passem 10 segundos e o campo não contenha esse valor o nosso teste falhará.

Listas de seleção

O próximo campo que vamos preencher é o estado civil do cliente. Vamos usar a *keyword* [Select From List By Label](#). Poderíamos usar a [Select From List By Index](#) ou [Select From List By Value](#). Nosso caso de teste vai ficar assim:

*** Test Cases ***

Fazer o cadastro de um novo cliente

Open Browser	https://estivalet.github.io/robot-static-testing-site/cadastro/index.html	chrome
Title Should Be	Cadastro de Clientes	
Input Text	id=nome_cliente	José da Silva
Click Element	xpath=//input[@value='masculino']	
Input Text	id=cep	21842640
Click Element	xpath=//button[text()='Pesquisar']	
Wait Until Keyword Succeeds	10s	200ms
Aguardar o valor "Rua Otávio de Farias" no elemento "id=rua" estar presente		
Select From List By Value	id=estado-civil	Casado(a)



*Muitos frameworks para desenvolvimento de aplicações web (Angular Material por exemplo) acabam criando elementos personalizados, e, muitas vezes, as caixas de seleção são implementadas com outras tags HTML que não a convencional **SELECT**. Se você estiver automatizando um sistema com esse tipo de elemento muito provavelmente as keywords “Select From List” não irão funcionar. Será necessário criar keywords personalizadas para interagir com esses elementos.*

Caixas de seleção

As caixas de seleção, ou *checkboxes*, são bem semelhantes ao caso dos *radio buttons* que vimos acima. Normalmente os *checkboxes* que pertencem a um mesmo grupo também têm **IDs** iguais. Esse é o nosso caso também no formulário de cadastro de clientes. Todos os *checkboxes* tem o id “interesses”.

Então teremos que usar um **xpath** para selecionar os interesses do cliente. Por exemplo, como vamos fazer para selecionar o interesse Testes Automatizados e Video Games? Para selecionar Testes Automatizados usamos o seguinte **xpath**:

```
//span[contains(text(),'Testes Automatizados')]/preceding::input[1]
```

Estamos dizendo que queremos localizar um elemento **SPAN** com o texto “Testes Automatizados” e que vamos pegar o primeiro input imediatamente antes desse **SPAN**, ou seja, vamos estar pegando o elemento **INPUT** que corresponde ao *checkbox* para essa opção. Vamos fazer o mesmo para selecionar o *checkbox* Video Games. Lembrando mais uma vez que é importante que o **xpath** criado retorne apenas uma ocorrência.

Para selecionar essas opções com o Robot Framework podemos usar novamente a *keyword* **Click Element**.

```
Click Element          xpath=//span[contains(text(),'Testes
Automatizados')]/preceding::input[1]

Click Element          xpath=//span[contains(text(),'Video
Games')]/preceding::input[1]
```

Finalizando o cadastro

Para finalizar a automação da página de cadastro de clientes basta apenas clicar no botão cadastrar após termos todos os campos obrigatórios preenchidos. Para validar o caso de teste vamos verificar se o sistema retorna a mensagem “**Cliente cadastrado com sucesso!**”. Dessa vez podemos usar a *keyword* **Wait** [Until Element Is Visible](#) e [Wait Until Page Contains](#) que irá aguardar até que o elemento que contém a mensagem seja exibida e que o texto de cadastro com sucesso esteja na tela respectivamente.

O nosso caso de teste completo deve ter ficado assim:

```
*** Settings ***

Library                      SeleniumLibrary

*** Test Cases ***

Fazer o cadastro de um novo cliente

    Open Browser              https://estivalet.github.io/robot-static-testing-
```

```

site/cadastro/index.html    chrome
Title Should Be             Cadastro de Clientes
Input Text                   id=nome_cliente    José da Silva
Input Text                   id=cpf          12345678900
Input Text                   id=dt nasc     01/04/2000
Click Element                xpath=//input[@value='masculino']
Input Text                   id=email       luiz@robotframework.org
Input Text                   id=cep        21842640
Click Element                xpath=//button[text()='Pesquisar']
Wait Until Keyword Succeeds  10s      200ms    Aguardar o valor "Rua Otávio de
Farias" no elemento "id=rua" estar presente
Input Text                   id=numero     347
Select From List By Value    id=estado-civil  Casado(a)
Click Element                xpath=//span[contains(text(),'Testes
Automatizados')]/preceding::input[1]
Click Element                xpath=//span[contains(text(),'Video
Games')]/preceding::input[1]
Click Element                id=cadastrar
Wait Until Element Is Visible id=msg
Wait Until Page Contains     Cliente cadastrado com sucesso!
Close Browser

```

*** Keywords ***

```

Aguardar o valor "${valor_esperado}" no elemento "${elemento}" estar presente
    ${valor_atual}=    Get Element Attribute    ${elemento}    value
    Should Be Equal As Strings    ${valor_atual}    ${valor_esperado}

```

Like

Be the first to like this.

